LAB101 Assignment

Type:	**Long Assignment**
Code:	**C.L.P0001.docx**
LOC:	**175**
Slot(s):	**3**

Title

Spell Checker.

Background Context

Many of us have horrible spelling and would get great practical use out of a spell-checker. In this assignment, you will write a simplified version of a spell checker. In the process, you will write several of your own string functions.

Program Specifications

The first line of the file will have a single positive integer, n (n ≤ 30000), representing the number of words in the file. The following n lines will contain one word each, in alphabetic order. All words will contain lowercase letters only and will be no longer than 29 letters long.

Function details:

1. Your program should first read in the dictionary into a two-dimensional character array (from dictionary.txt) and print a message to the screen stating that the dictionary has been loaded.
2. Then your program should prompt the user to enter a word, all lowercase.
3. If the word is in the dictionary, you should simply print out a message stating this.
4. If the word is NOT in the dictionary, then you should provide a list of possible suggestions (of correctly spelled words that the user might have been trying to spell) in alphabetical order.
5. Here are the criteria for whether or not a real word should end up on this list:
    - If either string is a substring of the other, and the lengths of the two strings are within 2 of one another.
    - If either string is a subsequence of the other and the lengths of the two strings are within 2 of one another. (This would get rid of the “car”, “camera” case, for example.)
    - If the strings are permutations of one the other. (Only try this test if the two strings are of the same length.)
    - If the matchscore between the two strings is less than 3. (Only try this test if the two strings are of the same length.)
6. Continue prompting the user to enter words until they decide that they don’t want to enter any more. (Thus, they will be forced to enter the first word, but after that, you’ll provide them a choice as to whether or not they want to enter another word to check

Expectation of User interface:

```
Welcome to the Spell Checker!
The dictionary has been loaded.
Please enter the word you would like checked.
flag
Great, flag is in the dictionary!
Would you like to enter another word? (yes/no)
yes
Please enter the word you would like checked:
peice
Here are the possible words you could have meant:

alice
beige
brice
deuce
fence
heine
hence
ice
juice
paine
peace
peach
peale
pease
pee
pence
perch
percy
perle
peste
pewee
pie
piece
place
poise
ponce
price
prick
pride
prime
prize
reich
seize
slice
spice
twice
voice
Would you like to enter another word? (yes/no)
yes
Please enter the word you would like checked.
independant
Here are the possible words you could have meant:
independent
Would you like to enter another word? (yes/no)
no
```

Guidelines

Note: Assume that all of the formal parameters for each of the functions below are lowercase alphabetic strings.

A string is a substring of another one if all of its letters appear contiguously (in the same order) in the second string. For example, “bound” is a substring of “homebound” and “ire” is a substring of “firefly”, but “car” is NOT a substring of “camera” because the letters ‘m’ and ‘e’ are in between the letters ‘a’ and ‘r’ in “camera.”

```
// Returns true if shortstr is a substring of longstr,
// and false otherwise.
int substring(char shortstr[], char longstr[]);
```

A string is a subsequence of another string if all the letters in the first string appear in the second string, in the same ordering, but not necessarily contiguously. For example, “car” is a subsequence of “camera”, since the letters ‘c’,’a’, and ‘r’ appear in that order (but not contiguously) in “camera.” Similarly, “hikes” is a subsequence of “chickens,” but “tale” is NOT a subsequence of “wallet” because the letter ‘t’ occurs AFTER the letter ‘a’ in “wallet.”

```
// Returns true if shortstr is a subsequence of longstr,
// and false otherwise.
int subsequence(char shortstr[], char longstr[]);
```

A permutation of letters is simply a different ordering of those letters. For example, “care” is a permutation of “race” and “spill” is a permutation of “pills,” but “apple” is NOT a permutation of “pale” because the letter ‘p’ appears only once instead of twice in “pale.” A natural consequence of this definition is that two strings can only be permutations of one another if they are both the same length.

```
// Returns true if string1 and string2 are permutatations
// of each other, false otherwise.
int permutation(char string1[], char string2[]);
```

When we are comparing two strings of equal length, we can define a term called “match score” which is simply the number of corresponding letters in which the two strings disagree. For example, the match score between “drink” and “blank” is 3 because the first three letters don’t match. The match score between “marker” and “master” is two because letters 3 (r vs. s) and 4 (k vs. t) do not match.

```
// Precondition: string1 and string2 are the same length
//               and only contain lowercase letters.
// Postcondition: returns the score of the match score
//                between these two strings.
int matchscore(char string1[],char string2[]);
```